

**University of Groningen**

## **Managing the complexity of variability in software product families**

Deelstra, Keimpe Sybren; Sinnema, Marco

**IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.**

*Document Version*

Publisher's PDF, also known as Version of record

*Publication date:*

2008

[Link to publication in University of Groningen/UMCG research database](#)

*Citation for published version (APA):*

Deelstra, K. S., & Sinnema, M. (2008). *Managing the complexity of variability in software product families*. s.n.

### **Copyright**

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

### **Take-down policy**

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

*Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.*

## Chapter 4 Case Studies

---

*As we discussed in the Introduction to this Thesis, we started our research by studying the problems and issues that arise during product derivation in practice. A large part of this study involved the case studies we performed at two industrial partners in the ConIPF project, i.e. Thales Nederland B.V., and Robert Bosch GmbH. Both companies are large and mature industrial organizations that mark two ends of a spectrum of product derivation; Robert Bosch produces thousands of medium-sized products per year, while Thales Nederland produces a small number of very large products. We present these case studies in terms of the product derivation framework we discussed in the previous chapter. In addition to these two case studies, we performed a third case study at Dacolian B.V. This case study is used to exemplify and validate our variability management framework COVAMOF.*

Based on	Section numbers
S. Deelstra, M. Sinnema, J. Bosch, Product Derivation in Software Product Families; A Case Study, Journal of Systems and Software, Vol. 74/2 pp. 173-194, January 2005.	Section 4.1 and Section 4.2
M. Sinnema, S. Deelstra, Industrial Validation of COVAMOF, Elsevier Journal of Systems and Software, Vol 81/4, pp. 584-600, 2007.	Section 4.3.1
M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, COVAMOF: A Framework for Modeling Variability in Software Product Families, Proceedings of the 3rd Software Product Line Conference (SPLC 2004), Springer Verlag Lecture Notes in Computer Science Vol. 3154 (LNCS 3154), pp. 197-213, August 2004.	Section 4.3.2

---

### 4.1. Thales Nederland B.V.

---

Thales Nederland B.V., the Netherlands, is a subsidiary of Thales S.A. in France and mainly develops Ground Based and Naval Systems in the defense area. Thales Naval Netherlands (TNNL), the Dutch division of the business group Naval, is organized in four Business Units, i.e. Radars & Sensors, Combat Systems, Integration & Logistic Support, and Operations. Our case study focused on software parts of the TACTICOS naval combat systems family produced by the Business Unit Combat Systems, more specifically, the Combat Management Systems.

#### **4.1.1. The Combat Management Systems product family**

A Combat Management System (CMS) is the prime subsystem of a TACTICOS (TACTical Information and COMmand System) Naval Combat System. Its main purpose is to integrate all weapons and sensors on naval vessels that range from fast patrol boats to frigates. The Combat Management System provides Command and Control and Combat Execution capabilities in the real world, as well as training in simulated worlds.

The asset base that is used to build Combat Management Systems, also referred to as the infrastructure, was first established in the 1990s. It provides a virtual machine (referred to as SigMA) as well as a publish/subscribe communication mechanism through a real time distributed database system (referred to as SPLICE). The common functionality is mainly concerned with handling the real time storage and transportation of message instances, the creation of virtual 'worlds' for training, replay of loggings and tests, as well as handling a duplicated LAN and dynamic reallocation of applications for battle damage resistance.

The asset base is now fairly stable, consists of approximately 1500 kLOC and contains both in-house developed and COTS (Commercial Off The Shelf) components. Each component version consists of the source, the binaries, and a fixed documentation associated with it. The descriptions of the components are stored in a hierarchical component repository, where the top-level denotes the functional component and the leaves the small-sized (10–120 kLOC) individual executables.

This component repository and surrounding management tasks play a supportive role within the organization. It mainly provides information and consultancy regarding the use of the components towards several primary processes, handles COTS purchase, component change and maintenance, as well as acceptance of new components to the repository.

Although the Business Unit Combat Systems is in the process of extending the scope of reuse for the Combat Management Systems product family to a software product line, the asset base captures functionality common to all Combat Management Systems and is treated as if it was an externally bought infrastructure. We therefore classify the Combat Management System family as a single product family with a platform as the scope of reuse (see also Section 3.2).

The products built on top of the infrastructure, i.e. the Combat Management Systems, are large and complex technical software systems. An average configuration (including the infrastructure) consists of approximately 37 main components and several MLOC. The infrastructure alone already requires setting several thousand lines of parameters, which often contain multiple parameter

settings per line. In the next Subsection, we discuss how the process of deriving the products is organized at TNNL Combat Systems.

#### 4.1.2. Derivation process

The derivation process at the Business Unit Combat Systems is highlighted in Figure 10 and discussed below.

**Initial phase.** Due to the size of the Combat Management Systems, and the large amount of similarities between configurations for similar types of ships, configuration selection is used to derive the initial product configurations (the thin arrow in Figure 10). To this purpose, the collected requirements are mapped onto an old configuration, whose characteristics best resemble the requirements at hand. In practice, this configuration is usually the most recently completed configuration of the Combat Management System for the same type of ship.

Using this old configuration, a complete specification of the software product is compiled. This specification is used in the rest of the product derivation process. First, the old configuration is modified to comply with the specification. To this purpose, the configuration is modified by re- and de-selecting components, adapting components and changing existing parameter settings.

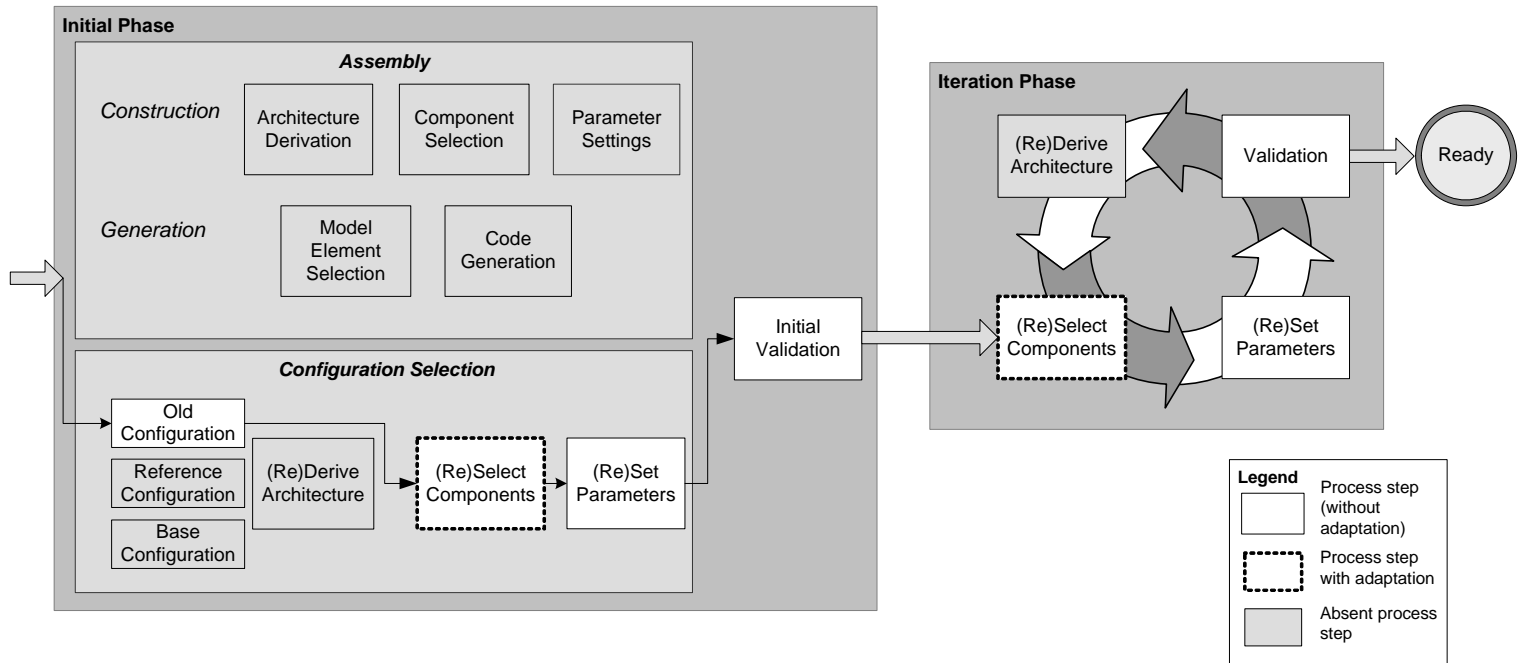
The architecture is not changed explicitly, but implicitly modified by the selection and adaptation activities. When all components and parameters are selected, adapted and set, the system is packaged and installed in a complete environment for the initial validation. If the configuration does not pass the initial validation, the derivation process enters the iteration phase.

**Iteration phase.** Until the product sufficiently adheres to the requirements, the configuration is modified in a number of iterations (the thick arrow in Figure 10), by re- and de-selecting components, adapting components and changing existing parameter settings.

**Adaptation.** During the (re)selection of components, both reactive evolution and product specific changes are applied when components are adapted. The decision whether component development or change will result in product-specific code or development that concerns the entire product family, is determined through Change Control Boards (CCB). Each product development project (which is an application engineering project) has its own CCB that determines whether a request for development for the product family will go to the component CCB (which is part of domain engineering).

The component CCB synchronizes the requests of different projects and decides whether and which of the requests will be honored. The changes are financed and performed by domain engineering. Components are also adapted through proactive evolution. Although all components were well documented at the moment they were initially developed, some of the documentation was not maintained completely when the components were changed. As a result, the derivation process at Combat Systems strongly depends on tacit knowledge. No formal descriptions are used during the derivation process.

The main distinct characteristics of the product families Robert Bosch GmbH are summarized in Table 4. When we relate the description of the product derivation process of TNNL Combat Systems to Section 3.3, evidently, the process at Combat Systems is an instance of the generic derivation process depicted in Figure 6.



**Figure 10. Product derivation at Combat Systems.** The customer requirements are mapped onto a closest matching old configuration, which is modified by re- and deselecting components and parameters. After the initial validation, the configuration is modified in a number of iterations, until the product is deemed ready.

---

## 4.2. Robert Bosch GmbH

---

Robert Bosch GmbH, Germany, was founded in 1886. Currently, it is a world-wide operating company that is active in the Automotive, Industrial, Consumer Electronics and Building Technology areas. Our case study focused on two business units, which, for reasons of confidentiality, we refer to as business unit A and B, respectively.

### 4.2.1. The product families

The systems produced by the business units consist of both hardware, i.e. sensors and actuators, and software. The main requirements for originate from regulations at various parts of the world, and different market segments (e.g. low-cost, mid-range or high-end).

Product family A captures both common and variable functionality of the product family members. Product family A is therefore classified as a family with the software product line as scope of reuse. The asset base of the product family B includes functionality that is common to many products in the family. It has a heartbeat of two months, which means every two months the set of shared artifacts is updated with customer specific functionality that is deemed useful for most other product family members. The scope of reuse for the product family of business unit B is therefore classified as a platform (see also Section 3.2).

While TNNL Combat Systems develops a few instances of the large and complex Combat Management Systems per year, the business units derive thousands of instances of systems per year. In the following Subsections, we discuss the derivation process of both business units.

### 4.2.2. Derivation process for business unit A

A system that is derived at business unit A, on average contains approximately 50 components, a few hundreds of compiler-switches and several thousands of runtime parameters. The process that is used for deriving these products is highlighted in Figure 11 and discussed below.

**Initial phase.** Starting from requirements engineering, business unit A uses two approaches in deriving an initial configuration of the product, i.e. one for lead products and one for secondary products:

- **Lead products.** For lead products, the first configuration of the product is constructed using the assembly approach (upper thin arrow in Figure 11). First, the architecture is derived from the product family architecture. In the next

step, the closest matching component implementations are selected from the repository and subsequently the parameters are set. If, during the selection of the components, no suitable component implementation can be found in the repository, the most suitable component is copied and adapted where necessary. If the required change involves a completely new concept, the component is developed from scratch (denoted by the dashed box around Component Selection in Figure 11).

- **Secondary products.** In case a similar product has been built before, the first version of the product is developed by using reference configurations (lower thin arrow in Figure 11). These reference configurations are lists of components for consistent configurations, and are typically configurations resulting from previous projects that are explicitly designated as reference point. The components from the reference configuration are replaced with more appropriate ones where necessary. As the reference configurations are lists of components, all parameters are still open and have to be set before the iteration step. Knowledge about valid parameter settings is available however, which restricts the range of variability for the parameter settings.

After the initial configuration is ready, the software product is packaged and installed on a test bench for calibration and validating the product with respect to completeness, correctness and consistency of the configuration. If the configuration passes the validation test, the product is deemed ready. If new customer requirements come in, or when the configuration does not pass validation, the derivation process enters the iteration phase.

**Iteration phase.** In the iteration phase, the initial configuration is modified in a number of iterations by reselecting components, adapting components, or changing parameters (the thick arrow in Figure 11). At business unit A, up to half of the time spent in deriving a product is consumed by this phase.

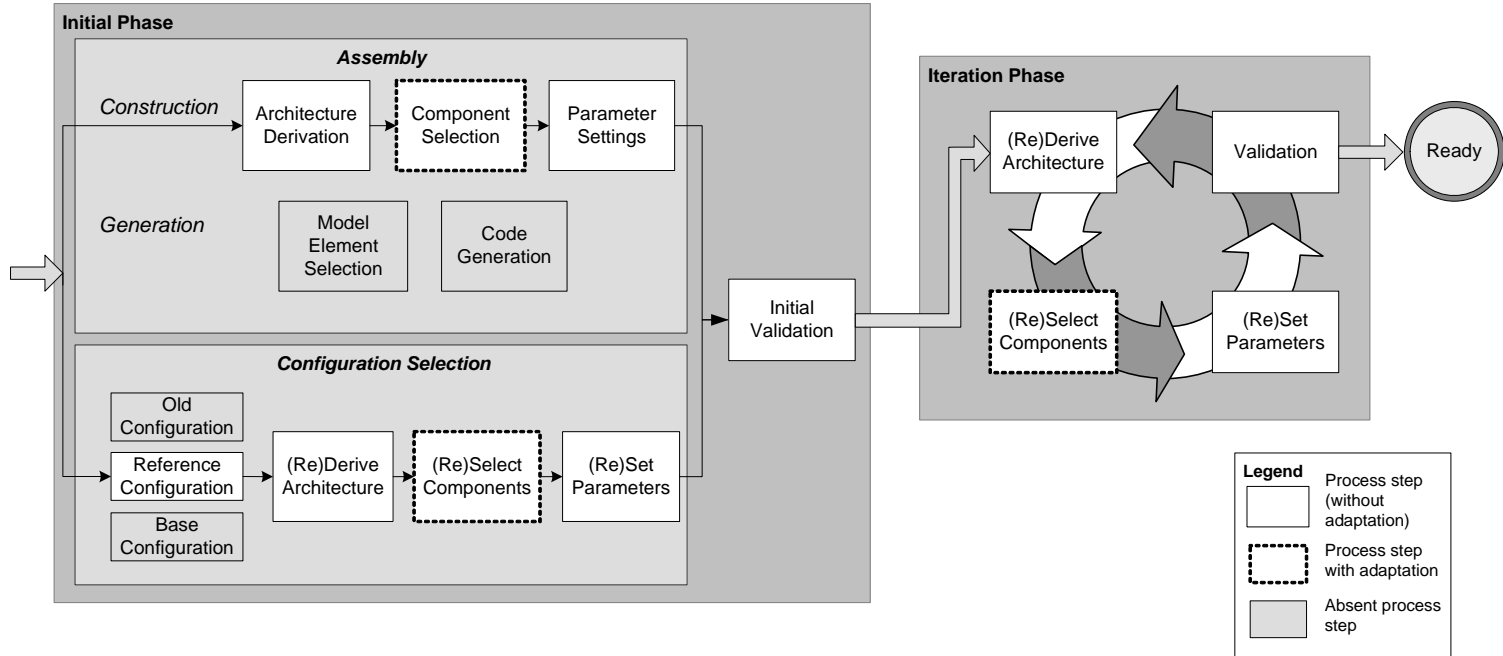
**Adaptation.** If the inconsistencies or new requirements cannot be solved by selecting a different component implementation, a new product family component implementation is developed through reactive evolution. This is achieved by copying and adapting an existing implementation, or developing one from scratch.

The component documentation for individual products often comprise thousands of pages, which are quite up-to-date. Furthermore, formal descriptions are available of the component interfaces. All other knowledge about the artifacts is available as tacit knowledge. The engineers indicate this tacit knowledge is still vital for the product derivation process.

The main distinct characteristics of the product family of business unit A are summarized in Table 4. When we relate the description of the product derivation



process of this business unit to Section 3.3, evidently, the process of unit A is an instance of the generic derivation process depicted in Figure 6.



**Figure 11 . Product derivation at business unit A. Lead products are derived through assembly, while secondary products are derived using reference configurations. The resulting initial configuration is modified in a number of iterations by reselecting components and resetting parameters. Requirements that cannot be handled using existing components are accommodated by reactively evolving the product family assets during the Component Selection and (Re)Select Components steps.**

### 4.2.3. Derivation process of business unit B

The product derivation process at business unit B is highlighted in Figure 12 and discussed below.

**Initial phase.** Each time a product needs to be derived from the product family, a project team is formed. This project team derives a product by assembly (the thin arrows in Figure 12). It copies the latest version of the shared components and selects the appropriate components from this copy. Thereafter, all parameters of the components are set to their initial values.

The configuration is then packaged and installed on a test bench for initial validation. If the configuration passes validation, it is deemed ready. Otherwise, the derivation process enters the iteration phase.

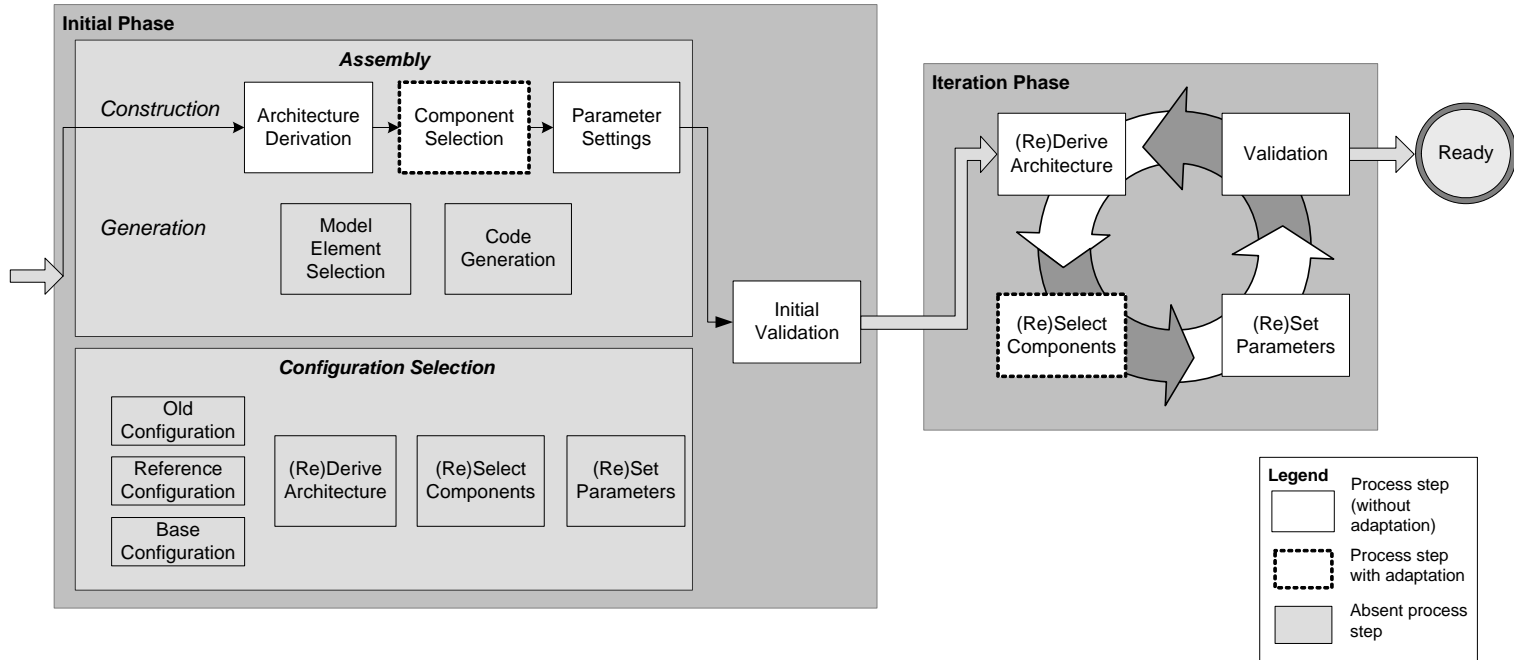
**Iteration phase.** When new requirements come in, or when inconsistencies arise during the validation process, components and parameters are reselected and changed in the iteration phase (the thick arrow in Figure 12), until the product is deemed finished. Similar to the situation at business unit A, the iteration phase at business unit B consumes up to half of the time spent in deriving a product.

**Adaptation.** When during the initial and the iteration phase the requirements for a product configuration cannot be handled by the existing product family assets, the changes are applied to the selected component copies, in other words, by product specific adaptation. These changes can therefore not be reused unless the new functionality is integrated into the shared artifacts at the heartbeat.

The main distinct characteristics of the product family of business unit B are summarized in Table 4. When we relate the description of the product derivation process above to Section 3.3, evidently, the process at business unit B is an instance of the generic derivation process depicted in Figure 6.

**Table 4. Case study characteristics. The main characteristics of the product families at Combat Systems, Thales Nederland B.V., and two business units at Robert Bosch GmbH.**

Product family	Scope of reuse	Initial derivation phase	Adaptation
TNNL Combat Systems	Platform	Old configuration	Reactive and product specific
Bosch A	Product line	Reference configuration and assembly	Reactive
Bosch B	Platform	Assembly	Product specific



**Figure 12. Product derivation at business unit B.** Products are derived by assembling an initial configuration. After the initial validation, the configuration is modified in a number of cycles in the iteration phase. Requirements that cannot be handled by existing product family assets are handled through product specific adaptation during the Components Selection and (Re)Select Components steps.

---

### 4.3. Dacolian Case Study

---

In addition to the case study that we used to identify problems and issues, we performed a case study at Dacolian B.V. to validate the variability management framework that we discuss in later parts of this thesis. Dacolian B.V. (see Dacolian Website) is a Dutch company that develops intellectual property software modules for Intelligent Traffic Systems (ITS). In this section, we describe the organization and its product family. Throughout this Thesis, we use examples from this case study to illustrate and validate our COVAMOF variability management framework.

#### 4.3.1. Organization and Product Family

Dacolian B.V. uses a product family called Intrada® to produce the software modules for Intelligent Traffic Systems (ITS). The product family artifacts of this family consist of approximately 11 million lines of code. Typical products consist of systems that deliver; based on real-time input images, abstract information on the actual contents of the images, e.g. the presence of traffic, vehicle type and brand, car registration numbers, etc. As such, Dacolian is worlds leading supplier of OEM software modules for intelligent traffic systems that utilize Automatic License Plate Recognition (ALPR), see Figure 13.



**Figure 13. Dacolian Worldwide.** This figure depicts an overview of the countries where Dacolian software is used for tolling, law-enforcement, and access control/parking.

Intrada modules can be found within the three major ALPR application areas, which are tolling, enforcement, and access control/parking (see also Figure 14). Examples of (open road) tolling systems around the world are the Highway 407ETR in Canada, the Costanera Norte in Chile, Nationwide Electronic Toll Collection in Taiwan, and several highways in the USA. Examples of the enforcement and parking application areas are the internationally well known Gatsometer cameras that are used for red light and speed enforcement (Intrada modules are embedded within these systems) and the parking systems at the Frankfurt International airport system of VMT Düssel, respectively. Each of these application areas provides its own distinct set of performance requirements and image characteristics resulting in a different configuration of Intrada OEM modules.



**Figure 14. Automatic License Plate Recognition.** One of the application areas of automatic license plate recognition of Dacolian B.V. is tolling on highways.

4.3.2. Intrada Product Family Architecture

Dacolian maintains a hierarchical product-family architecture, of which we present the logical view in Figure 15. Dacolian furthermore maintains in-house developed reusable components that are used for product construction. Many of the components are frameworks. Examples of these frameworks are a common interface to a number of real and virtual frame grabbers (Image Intake component in Figure 15) and a framework providing license specialization and enforcement (License Manager component in Figure 15). The infrastructure also contains special designed tooling for Model Driven Architecture (MDA) based code generation, software for module calibration, dedicated scripts and tooling for product, feature and component testing, and large data repositories. These assets capture functionality common to either all Intrada products or a subset of Intrada products.

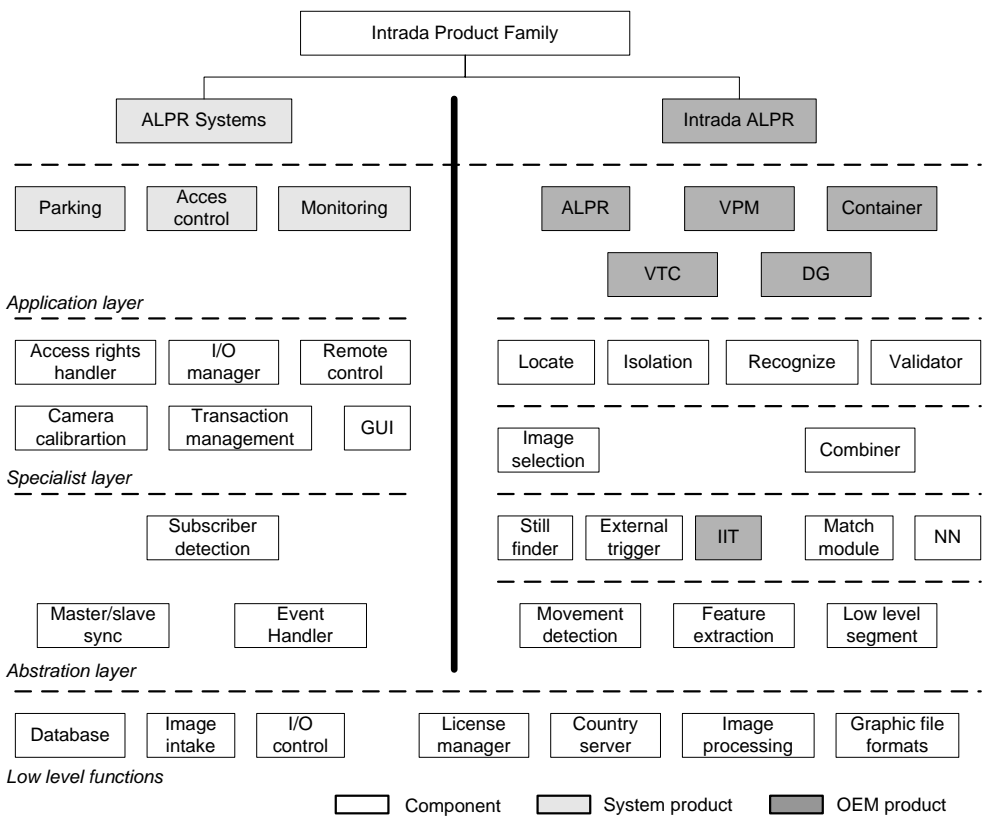


Figure 15. Logical view on the Hierarchical Intrada Product Family.

The responsibilities of the reusable components in Figure 15 are denoted by the layers in the logical view. Each component incorporates variability and uses a

selection of the components in a lower abstraction level. The thick solid vertical line represents the separation between the two branches within the Intrada Product Family, i.e. the *Intrada Systems* and *Intrada ALPR* families.

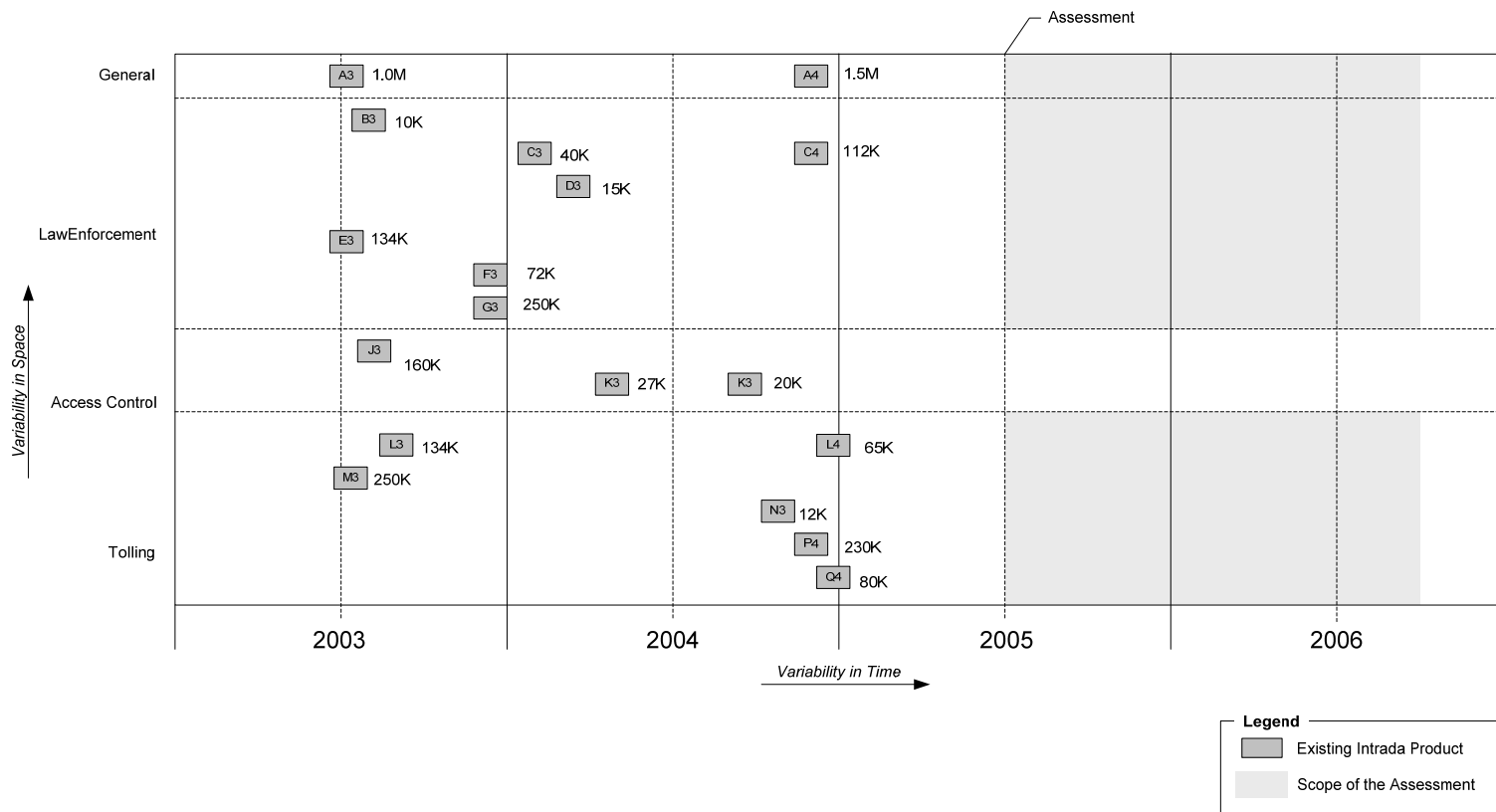
The low level functions layer provides basic functionality, hides hardware detail and gives platform independence. The abstraction layer adds semantic information to the raw image data. The specialist layer contains core components of Dacolian's Intrada product family. The blocks in this layer perform the complex calculations. The top layer defines the various products. The grey shaded blocks indicate products that are sold by Dacolian B.V. The OEM-products (dark grey) have no interaction with hardware components, whereas the System products (light grey) like parking, access control, and monitoring include components that communicate directly with the real world.

For products at the upper layer, binding is typically done by configuration and license files at startup time for the products at the upper layer. Lower layers use binding mechanisms that bind before deployment. Typical examples are MDA code generation, pre-compiler directives and Make dependencies. Binding at link time is almost not applied in the Intrada product family, and Make dependencies only specify variation at compile time.

#### **4.3.3. Evolution**

The Intrada ALPR product family evolved over the past seven years. In Figure 16, we depict the products that were sold during the two years prior to the case study. In that period, the Intrada ALPR product family contained four different products groups (the space dimension), and all product groups evolved over several generations (the time dimension). At the time of the case study, the most recent version of the Intrada ALPR family was 4.0.0, but several customers still had products in the field that had been built with older releases of the Intrada ALPR family.





**Figure 16. Intrada ALPR product evolution.** This figure depicts different products of the Intrada ALPR product family. The graph shows that, the Intrada ALPR product family evolved in both space (increasing number of different products), as well as time (new versions of products).